Compiling Linguistic Constraints into Finite State Automata

Mathieu Constant, Denis Maurel

Compiling Linguistic Constraints into Finite State Automata

Matthieu Constant[1] and Denis Maurel[2]

[1] Université de Marne-la-Vallée, Institut Gaspard Monge, France
[2] Université François-Rabelais de Tours, Laboratoire d'Informatique, France

Abstract. This paper deals with linguistic constraints encoded in the form of (binary) tables, generally called lexicon-grammar tables. We describe a unified method to compile sets of tables of linguistic constraints into Finite State Automata. This method has been practically implemented in the linguistic platform Unitex.

1 Motivation

Finite State Models have been intensively used in Natural Language Processing [13]. Nevertheless, because of the complexity of languages, it is often more convenient for linguists to describe linguistic constraints with simpler and more ergonomic representations. For instance, simple regular expressions are sometimes used to express morphological rules [6], inflected forms of dictionaries are preferred to be written in a textual form [3] and syntactic constraints depending on lexicon are represented in the form of binary matrices [4]. Finite State linguistic phenomena are sometimes described with more powerful and more compact formalisms such as (weighted) context-free grammars [10] and recursive transition networks[5]. These representations are then compiled into Finite State Automata or Transducers in order to optimize processing.

This paper deals with linguistic constraints encoded in the form of (binary) tables made of rows and columns, generally called lexicon-grammar tables. A row of such table corresponds to the formal description of the lexical and syntactic properties accepted by a lexical item. Each column corresponds to a property. At the intersection of a row and a column, the encoded value indicates whether or not a lexical entry (row) accepts a property (column)[1]. In this paper, we will describe a unified method to compile sets of tables of linguistic constraints into Finite State Automata. We will also show how it has been practically implemented in the linguistic platform Unitex [11].

2 State-of-the-Art

The first idea of combining binary matrices and automata was pointed out in [7], but the first compilation method has been found in [12] and has been implemented in the linguistic platforms INTEX [14] and Unitex [11]. It was limited

[1] Usually, symbol `+` stands for `True` and symbol `-` stands for `False`.

O.H. Ibarra and H.-C. Yen (Eds.): CIAA 2006, LNCS 4094, pp. 242–252, 2006.

to systems of constraints encoded in one table such as the ones in [4]. It used hand-built parameterized reference automata, representing the sets of the possible syntactic constructions where can enter a fictive lexical entry accepting all properties of the table. Each path is parameterized by one or several parameters that refer to properties that correspond to syntactic constructions (e.g. `Prep Det Noun`[2]) or lexical information (e.g. if the constituent `Prep` accepts the lexical value *in*). The compilation process consists, for each lexical entry (or raw), in resolving the parameters according to the encoding in the tables. For instance, a false value at a given column indicates that the transitions labeled with the parameter associated with the column, must be removed. A true value indicates that these transitions must be made epsilon-transitions. Then, a specific automaton is constructed for each lexical entry. The automaton representing all described phenomena is simply the union of all constructed automata. It is then optimized by a deterministic minimization operation for text processing efficiency.

Several linguistic studies have shown that it is sometimes more convenient to encode constraints of a same linguistic phenomena into systems of multiple tables because some properties can be factorized in different tables to avoid encoding duplication [7,1]. In this case, Roche's compilation does not work because it does not handle multiple tables. [8] implemented an algorithm compiling systems of multiple tables of specific constraints. These constraints were limited to very local constraints. Tables described the restrictions on the combinations of pairs of lexical elements in sequences where both elements occur consecutively (or sometimes with a grammatical word in between). For instance, for French time expressions, sequence *milieu de matin* (middle of morning) is forbidden while sequence *milieu d'après-midi* (middle of afternoon) is accepted. A schemata automaton is used to represent all possible patterns for a type of expressions. This automaton also recognizes bad sequences because it does not take lexical restrictions into account. All forbidden sequences encoded in the tables are put in an automaton that is then applied using the failure algorithm [9] that cuts all forbidden paths in the schemata automaton. [2] proposed an algorithm with no restrictions on the constraints; constraints were represented in relational systems of tables. The algorithm consisted in directly constructing the automaton that recognizes accepted sequences, by using a parameterized reference automaton with parameters resembling Roche's ones. Nevertheless, the complexity of the construction of the parameterized automaton could grow very fast with the number of tables. For instance, it is not well adapted to Maurel's time expressions.

In this paper, we present a unified algorithm for compiling systems of tables of constraints with no restrictions on the type of constraints.

3 Set of Constraints and Parameterized Automaton

This section focuses on the general description of inputs of our algorithm, that are a set of linguistic constraints and a parameterized schemata. They are respectively described in section 3.1 and in section 3.2.

[2] `Prep Det Noun` stands the construction `preposition determiner noun`

3.1 Sets of Linguistic Constraints

A syntactic construction is a sequence of syntactic symbols (and sometimes of lexical symbols): for instance, the syntactic construction `N0 V N1` is composed of a noun phrase (`N0`) followed by a verb (`V`) and then another noun phrase (`N1`). Each syntactic symbol have a set of possible lexical realizations, e.g. `V` could be `eat` or `walk`. Though, syntactic constructions have lexical restrictions; their acceptability can depend on the lexical realizations of a syntactic element. For example, the transitive verb `eat` can enter the constructions `N0 V N1`, while the intransitive verb `walk` cannot[3]:

```
John is eating an apple.
*John is walking an apple.
```

Such constraint is called a one-dimensional constraint because it depends on only one element (the verb).

There can also exist lexical restrictions on the combination of two syntactic elements in the context of a construction. For instance, in the construction `N0 V N1 Prep N2`[4], there exist lexical constraints on the pair `(V,Prep)`: pairs `(receive,from)` and `(give,to)` are acceptable, while `(receive,to)` and `(give,from)` are forbidden as it is shown in the sentences below.

```
John received a present (*to+from) Mary.
John gave a present (to+*from) Mary.
```

Such constraint is called a two-dimensional constraint because it depends on the combination of two elements (the pair verb-preposition).

Practically, a given constraint is not only limited to a single construction, but also a set of equivalent constructions. For instance, the constraint on the pair `(V,Prep)` in the example above is available as well for the equivalent interrogative construction :

```
Who received a present (*to+from) Mary ?
```

Moreover, linguistic constraints can also restrict the combination of more syntactic elements cooccurring in a same construction. Theoretically, such constraints can be decomposed into elementary constraints that are one-dimensional and two-dimensional ones, all related with logical AND operators. For example, the acceptability of frozen constructions of the form `N0 be Prep N Prep1 N1`, can depend on the lexical combination of `Prep`, `N` and `Prep1` such as in:

```
The text is (in+*on) contradiction (with+*to) the law.
```

Verifying if this constraint is valid is equivalent to checking if elements `in` and `contradiction` can cooccur in this context and if `contradiction` and `with`

[3] In linguistic examples, the symbol `*` is the forbidden symbol and symbol `+` is the disjonction symbol.

[4] `N0`, `N1` and `N2` are noun phrases, `V` is a verb and `Prep` a preposition.

cooccur. Thus, in the next sections, we will consider that there exist only one-dimensional and two-dimensional constraints. One-dimensional ones are encoded in the form of binary vectors, each element corresponding to a lexical value; two-dimensional lexical constraints are encoded in the form of binary matrices, encoding the restrictions on the combination of pairs of lexical values.

Examples of such representations are given in figure 1. The binary representations describe lexical constraints on geographical names. Such names can enter two constructions `Detc Npr Nc` (labeled `NN`) and `Detc Nc of Npr` (labeled `NPN`), where `Detc` is a definite determiner (e.g. `the`), `Npr` is a proper name such as `Adriatic`, `Marmara`, `Paris`... and `Nc` is a location noun classifier like `city`, `sea`... Figure 1(a) presents two-dimensional constraints between lexical realizations of `Nc` and `Npr` (`sea` and `Adriatic`); figure 1(b) and figure 1(c) present one-dimensional constraints depending on `Npr`, indicating whether or not it can enter constructions `NPN` (`city` of `Paris`) or `NN` (`Adriatic sea`).

	Adriatic	Marmara	Paris
city	-	-	+
sea	+	+	-

(a) Names-Classifiers

	Adriatic	Marmara	Paris
NPN	-	+	+

(b) NPN constraint

	Adriatic	Marmara	Paris
NN	+	+	-

(c) NN constraint

Fig. 1. One- and two-dimensional constraints

3.2 Parameterized Schemata Automaton

A parameterized schemata automaton is a hand-built acyclic automaton that explicitely represents all possible syntactic realizations that the studied linguistic phenomenon can have. It is used as a basis to build an automaton representing all accepted constructions of this phenomenon, taking encoded lexical restrictions into account. Each path represents a possible construction. Labels of this automaton are either lexical or syntactic elements, or parameters. Syntactic elements that may cause lexical constraints in the construction are marked as parameters. They are called syntactic parameters. Such parameters are denoted with the name of the syntactic element preceded by symbol `@`: for instance, `@X` is the parameter associated with the syntactic symbol `X`. Sets of constructions (i.e. sets of paths) can also be parameterized because their acceptability may depend on the lexical realizations of some syntactic "parameterized" elements. We call them construction parameters. They are denoted with the label assigned to the set of constructions, preceded and followed by symbol `@`: for example, `@P@` is the parameter associated with the constructions labeled `P`. An example of such automaton is given in figure 2: it consists of the parameterized schemata automaton used for geographical names. `@Nc@` and `@Npr@` are syntactic parameters; `@NN@` and `@NPN@` are construction parameters.

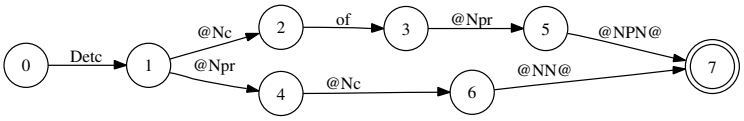

Fig. 2. A parameterized schemata automaton

A pair of parameters defines a set of lexical constraints. A pair composed of a syntactic parameter `@X` and a construction parameter `@P@` defines all 1-dimensional lexical constraints depending on the syntactic element `X` in the context of the constructions parameterized with `P`[5]. A pair composed of two syntactic parameters (`@X` and `@Y`) defines all 2-dimensional lexical constraints for combining the syntactic elements `X` and `Y` in the studied linguistic phenomenon[6].

4 Algorithm

Our algorithm for compiling a set of constraints into a finite state automaton (A^+) is based on the use of a parameterized schemata automaton (Ap). It consists of 3 steps:

1. building the automaton of all possible sequences (A) from Ap;
2. constructing the forbidden sequence automaton A^-, from Ap and the sets of different constraints;
3. constructing automaton A^+ defined by $L(A^+) = L(A) - L(A^-)$, where $L(A)$ stands for the language recognized by automaton A.

One can wonder why the desired automaton A^+ is not directly constructed from Ap. It is simply due to the fact that adding a new path requires checking all constraints it undergoes. In case of complex systems like the one proposed by [8] for date adverbials, the cost would be very important. At worse, the construction process would have exponential complexity. The idea to build first the automaton of forbidden constructions is because a path is invalid if it undergoes only one forbidden constraint. Its construction is then linear with the number of constraints[7]. The construction of the automaton of accepted sequences is simply implemented using an intersection-type algorithm.

4.1 Construction of the Automata of All Possible Constructions

Automaton A (step 1) is built by replacing the syntactic parameters of Ap by their actual associated symbols (standing for word classes), and replacing construction parameters by ϵ symbol. The automaton produced for geographical

[5] Path 5-7 comes from Fig. 1 (b) and path 6-7 comes from Fig. 1 (c).
[6] Paths 1-2-3-5 and 1-4-6 come from Fig. 1 (a).
[7] One should remark that the determinization operation computed after the automata construction is theoretically exponential in complexity. Nevertheless, it has been observed that, practically, it is very often not the case for natural language automata.

names is given in figure 3. Symbol <E> stands for ϵ symbol. Note that syntactic symbols are references to classes of words; they are also represented with automata. These automata are automatically built from the set of lexical realizations used in the set of constraints. Automaton A is obtained by replacing all syntactic symbols by their actual word class and then computing some optimization operations as it is shown in figure 4.

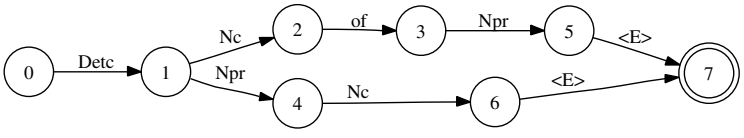

Fig. 3. A schemata automaton

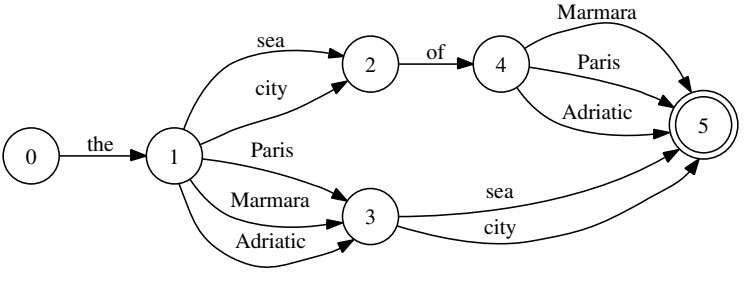

Fig. 4. A lexicalized schemata automaton

4.2 Construction of the Automaton of Forbidden Syntactic Forms

The construction of A^- consists in building, automatically from Ap, a parameterized automaton for each pair of parameters (X,Y) that undergoes lexical restrictions; and then in lexicalizing this automaton according to the restrictions encoded in the corresponding vectors or tables.

The construction of the pair-specific parameterized schemata automaton, called $Ap(X, Y)$, consists in keeping only paths of Ap where X and Y cooccur. Marking such paths is based on an automaton transversal-type algorithm: for each parameter of the pair, we mark the states and transitions of Ap, which can be reached from transitions labeled by the parameter, or from which such a transition can be reached. Then, $Ap(X, Y)$ is obtained by keeping states and transitions of Ap marked for both parameters. Finally, other parameters are either replaced by an ϵ (<E>) if they are construction parameters, or replaced by their associated syntactic symbol (refering to a word class) if they are syntactic parameters. An example of such automaton for the pair (Nc,Npr) is given in figure 5.

The construction of the automaton of forbidden syntactic forms is then based on the lexicalization of such pair-specific parameterized schemata automata. Such an automaton is associated with a binary vector or a binary table. In case

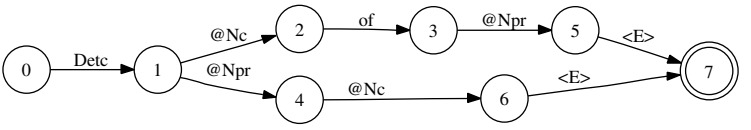

Fig. 5. the (Nc,Npr) parameterized schemata automaton

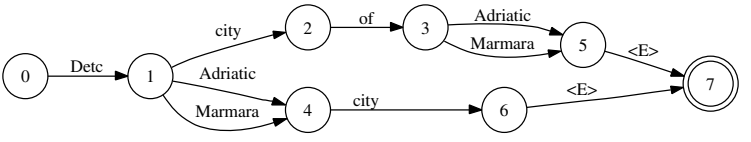

Fig. 6. the (Nc,Npr) lexicalized schemata automaton for entry `city`

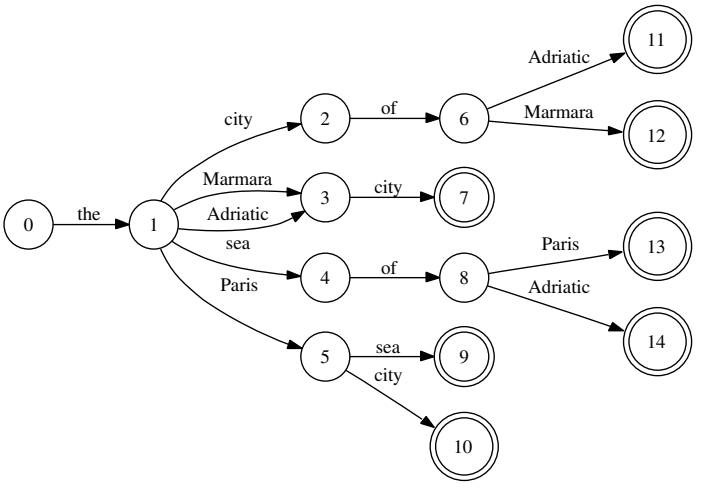

Fig. 7. Automaton of forbidden constructions for geographic names

of a binary vector associated with a parameter pair `(@X,@Y@)`, for each lexical entry x that are lexical realization of X, a new automaton is created from $Ap(X, Y)$. Parameters $@X$ are replaced by x. If the constructions labeled Y are not accepted, parameters $@Y@$ are replaced by ϵ (`<E>`). If they are, transitions labeled by $@Y@$ are removed. In case of a binary table associated with a parameter pair `(@X,@Y)`, for each lexical realization x of X, a new automaton is created from $Ap(X, Y)$. Transitions $(q, @Y, p)$, where p and q are states of the new automaton, are removed. For each lexical realization y of Y, if the combination between x and y is forbidden, a new transition (q, y, p) is added. Figure 6 shows an example of a lexicalized automaton specific to the pair `(Nc,Npr)` for the lexical entry `city` in the case of geographic names.

All obtained lexicalized automata are then unioned; all syntactic symbols are replaced by their actual automata; and finally, an optimization operation

is computed (useless state removal and determinization). The global lexicalized automaton of forbidden constructions for geographic names is given in figure 7.

4.3 Construction of the Automaton of Possible Syntactic Forms

Given deterministic automata A and A^-, it is then possible to compute automaton A^+ representing all possible constructions, taking all lexical restrictions into account. The process consists in computing the automaton that recognizes the language $L(A^+)$ defined such that $L(A^+) = L(A) - L(A^-)$. It implements a variant of the standard algorithm for computing the intersection between two automata. The automaton obtained for geographical names is given in figure 8.

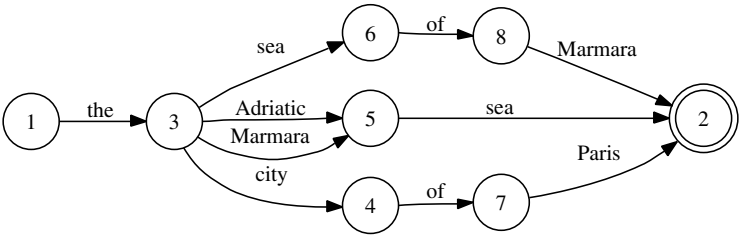

Fig. 8. Automaton of accepted constructions for geographic names

5 Implementation

Our algorithm has been implemented in C in Unitex, a GPL linguistic platform [11]. The implementation was eased by the use of some modules, data structures on finite state automata and common operations on them, already implemented in Unitex. Parameterized schemata automata can be drawn with a graph editor included in the platform. Unitex automata are recursive automata (automata that can call other automata) that are equivalent to Recursive Transition Networks (RTN) [15]. Therefore, syntactic symbols are simply calls to automata that represents their associated word classes. Automata can be unioned by simply creating an automaton that concurrently call all of them. There also exists a "Flatten" operation that computes the equivalent finite-state automaton of a given RTN (when recognizing a regular language).

Besides, it is often more convenient for linguists to have different parameter-pair constraints encoded in a same table in order to have a better view of the studied linguistic phenomenon[8]. Moreover, tables are not always binary: they can contain lexical values. For example, the example table and vectors (figure 1) are practically gathered in one table as in figure 9. We therefore implemented a module that transforms real tables into several binary vectors or tables.

[8] As it has been shown in [2], elementary constraints can be also gathered in several real tables.

Nc	Npr	Detc Nc of Npr	Detc Npr Nc
city	Paris	+	-
sea	Adriatic	-	+
sea	Marmara	+	+

Fig. 9. A real table of linguistic constraints

6 Evaluation and Discussion

We tested[9] our software with some lexicon-grammar tables describing constraints on French geographical locative phrases [1] and French time adverbials [8]. The test results are gathered in table 1. Samples of both systems are respectively given in figure 10 and figure 11.

Table 1. Test results

type	# of lexical constraints	# of parameter pairs	# of A^- states	# of A^- transitions	# of A^+ states	# of A^+ transitions	Compiling time
Loc	1,420	15	47,822	81,760	253	782	29 s
Time	818	5	2,868	27,280	100	694	2 s

Our unified algorithm has the advantage of working for all different types of systems of lexicon-grammar tables. Although compiling times are reasonable (cf. table 1), our algorithm is not always the most efficient one. For instance, the conversion of simple systems of relational tables like geographical phrases ones is faster using a process based on Roche's algorithm preceded by a merge operation on the related tables: 4s instead of 29s for our converter. Comparison is not feasible for time adverbials because Roche's algorithm and its extensions do not work in that case.

Constructing automaton A^- is the main factor for slowing down the process because A^- tends to be much bigger than the final A^+ (cf. table 1). Roche's algorithm and its extension (to systems with multiple tables) directly deal with A^+. Maurel's algorithm simply constructs the automaton recognizing forbidden subsequences, which is clearly a smaller automaton than A^- (542 states and 4191 transitions for 50 tables or parameter pairs [8]). The determinization of these automata makes the difference clearer because of the exponential complexity of this operation. Nevertheless, we consider that this relative lack of efficiency is not really important because compiling can be done once for all before applying

[9] Pentium III, 1.6 GHz, 512 Mb RAM.

Detc pluriel	Nc	Prep	Det	Npr	LE Nc de Npr	LE Nc Npr	dans Det Npr	à Det Npr
+	île	-	-	Aléoutiennes	-	+	+	+
-	île	de	la	Ascension	-	-	-	-
+	île	-	-	Bahamas	-	+	+	+
+	île	de	les	Baléares	-	+	+	+
-	île	de	la	Barbade	-	-	-	+
-	île	de	-	Beauté	+	-	-	-
+	île	-	-	Bermudes	-	+	+	+

(a) Islands

Nc	Prep	Det	Npr	LE Nc de Npr	en Npr	dans Det Npr
département	de	l'	Ain	-	-	+
département	de	l'	Aisne	-	-	+
département	de	l'	Allier	-	-	+
département	de	les	Alpes-de-Haute-Provence	-	+	+
département	de	les	Hautes-Alpes	-	-	+
département	de	les	Alpes-Maritimes	-	-	+
département	de	l'	Ardèche	+	+	+

(b) French Departments

Loc = :à	Loc =: dans	Loc =:sur	Loc =: en	Loc =:E	Nc
-	+	+	-	-	département
+	+	+	-	-	île

(c) Prepositional distribution

Fig. 10. Locative geographical phrases

	matin (morning)	midi (12 a.m.)	soir (evening)
aujourd'hui (today)	-	+	-
hier (yesterday)	+	+	+
demain (tomorrow)	+	+	+

	aujourd'hui (today)	hier (yesterday)	demain (tomorrow)
à	-	-	-
après	+	+	-
d'ici	+	-	+

Fig. 11. Time adverbials

the compiled automata on different texts. We are more interested in the fact that the algorithm works for all types of systems.

7 Conclusion

In this paper, we have shown that linguistic constraints encoded in the form of (binary) tables can be compiled into finite state automata. We have describe a unified method, implemented in the linguistic platform Unitex, in three steps: building the automaton of all possible sequences, the forbidden sequence automaton and the resulting automaton.

References

1. Constant, Matthieu. 2002. On the Analysis of Locative Phrases with Graphs and Lexicon-Grammar: the Classifier/Proper Noun Pairing. In Advances in Natural Language Processing, Proceedings of PorTAL, Lecture Notes in Artificial Intelligence (LNAI) 2389, Berlin: Springer, 33-42.
2. Constant, Matthieu. 2003. Converting Linguistic Systems of Relational Matrices into Finite-State Transducers. Proceedings of the EACL Workshop on Finite-State Methods in Natural Language Processing, Budapest, 75-82
3. Courtois, Blandine. 1990. Un système de dictionnaires électroniques pour les mots simples du français. Langue Française 87, Paris: Larousse
4. Gross, Maurice. 1984. Lexicon-Grammar and the Syntactic Analysis of French. In Proceedings of the 10 th International Conference on Computational Linguistics (COLING'84), Stanford, California.
5. Gross, Maurice. 1997. The Construction of Local Grammars. In Finite-State Language Processing, E. Roche & Y. Schabès (eds.), Language, Speech, and Communication, Cambridge, Mass.: MIT Press, 329-354.
6. Karttunen, Lauri, Jean-Pierre Chanod, Gregory Grefenstette and Anne Schiller. 1996. Regular Expressions for Language Engineering. CUP Journals: Natural Language Engineering 2 (4), Cambridge University Press, 305-328.
7. Maurel, Denis. 1989. Reconnaissance de séquences de mots par automates, Adverbes de dates du français. PhD. thesis, Paris: Université Paris 7.
8. Maurel, Denis 1996. Building automaton on Schemata and Acceptability Tables. First Workshop on Implementing automata (WIA'96), London, Ontario, 29-31 août, in LNCS 1260, 72-86.
9. Mohri, Mehryar. 1994. Syntactic Analysis by Local Grammars Automata: an Efficient Algorithm. In Proceedings of COMPLEX 94, Budapest, Hungary.
10. Mohri, Mehryar and Fernando C. N. Pereira. 1998. Dynamic compilation of weighted context-free grammars. 36th Annual Meeting of the ACL and 17th International Conference on Computational Linguistics, Vol. 2, 891-897.
11. Paumier, Sébastien. 2003. De la reconnaissance de formes linguistiques à l'analyse syntaxique. PhD. thesis, Université de Marne-la-Vallée.
12. Roche, Emmanuel. 1993. Une représentation par automate fini des textes et des propriétés transformationnelles des verbes. Lingvisticae Investigationes XVII:1Amsterdam/Philadelphia, John Benjamins, 189-222.
13. Roche, Emmanuel and Yves Schabes. 1997. Finite-State Language Processing, Cambridge, Mass./London, The MIT Press, 241-281.
14. Silberztein, Max D. . 1999. INTEX: a Finite State Transducer Toolbox". Theorical computer science. Vol. 231:1, 33-46.
15. Woods, William A.. 1970. Transition Network Grammars for Natural Language Analysis, Communications of the ACM, 13:10